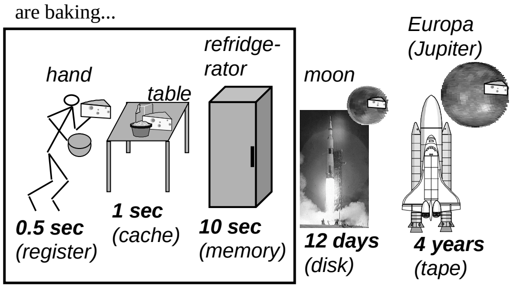
Memory Hierarchy and Cache
Ch 4-5

Memory Hierarchy
Main Memory
Cache
Implementation

Teemu's Cheesecake

Fig. 4.1

Register, on-chip cache, memory, disk, and tape speeds relative to times locating cheese for the cheese cake you are baking...



Goal (4)

- I want my memory lightning fast
- I want my memory to be gigantic in size

- Register access viewpoint: cache
  - data access as fast as HW register
  - data size as large as memory HW solution
- Memory access viewpoint virtual memory
  - data access as fast as memory
  - data size as large as disk HW help for SW solution

Memory Hierarchy (5)

Fig. 4.1

- Most often needed data is kept close
- Access to small data sets can be made fast
  - simpler circuits
- Faster is more expensive
- Large can be bigger and cheaper (per byte)

Memory Hierarchy

up:	smaller, faster, more expensive, more frequent access
down:	bigger, slower, less expensive, less frequent access

Principle of locality (7)

(paikallisuus)

- In any given time period, memory references occur only to a <u>small subset</u> of the whole address space Fig. 4.2
- The reason why memory hierarchies work

Prob (small data set) = 99%
Prob (the rest) = 1%

Cost (small data set) = 2 µs
Cost (the rest) = 20 µs

Aver cost 99% * 2 µs + 1% * 20 µs = 2.2 µs

- Average cost is close to the cost of small data set
- How to determine that small data set?
- How to keep track of it?

Principle of locality (5)

- In any given time period, memory references occur only to a <u>small subset</u> of the whole address space (paikallisuus)
- <u>Temporal locality:</u> it is likely that a data item referenced a short time ago will be referenced again soon (ajallinen paikallisuus)
- <u>Spatial locality:</u> it is likely that a data items close to the one referenced a short time ago will be referenced soon (alueellinen paikallisuus)

memory 2511: | 345 | 23 | 71 | 8 | 305 | 63 | 91 | 2 |

Memory

- Random access semiconductor memory
  - give address & control, read/write data
- ROM, PROMS
  - system startup memory, BIOS (Basic Input/Output System)
    - load and execute OS at boot
  - also random access
- RAM
  - "normal" memory accessible by CPU

Table 5.1 (Table 4.2 [Stall99])

RAM

E.g., $0.12 / MB (year 2001)?

- Dynamic RAM, DRAM
  - simpler, slower, denser, bigger (bytes per chip)
  - main memory?
  - periodic refreshing required
  - refresh required after read

E.g., 60 ns access

- Static RAM, SRAM

E.g., $0.50 / MB (year 2001)?

  - more complex (more chip area/byte), faster, smaller (bytes per chip)
  - cache?
  - no periodic refreshing needed
  - data remains until power is lost

E.g., 5 ns access?

DRAM Access

- 16 Mb DRAM
  - 4 bit data items
  - 4M data elements, 2K * 2K square
  - Address 22 bits
    - row access select (RAS)
    - column access select (CAS)
    - interleaved on 11 address pins
- Simultaneous access to many 16Mb memory chips to access larger data items
  - Access 8 bit words in parallel? Need 8 chips.

Fig. 5.3 (Fig. 4.4 [Stal99])

Fig. 5.4 (b) (Fig. 4.5 (b) [Stal99])

Fig. 5.5 (Fig. 4.6 [Stal99])

SDRAM (Synchronous DRAM)

- 16 bits in parallel
  - access 4 DRAMs (4 bits each) in parallel
- CPU clock synchronizes also the bus
  - not by separate clock for the bus
  - CPU knows how longs it takes make a reference – it can do other work while waiting
- Faster than plain DRAM
- Current main memory technology (year 2001)

E.g., $0.11 / MB (year 2001)

RDRAM (RambusDRAM)

- New technology, works with fast memory bus
  - expensive

E.g., $0.40 / MB (year 2001)?

- Faster transfer rate than with SDRAM

E.g., 1.6 GB/sec vs. 200 MB/sec (?)

- Faster access than SDRAM

E.g., 38 ns vs. 44 ns

- Fast internal Rambus channel (800 MHz)
- Rambus memory controller connects to bus
- Speed slows down with many memory modules
  - serially connected on Rambus channel
  - not good for servers with 1 GB memory (for now!)
- 5% of memory chips (year 2000), 12% (2005)?

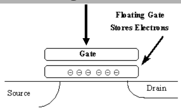
Flash memory

- Original invention
  - Fujio Masuoka, Toshiba Corp., 1984
  - non-volatile, data remains with power off
  - slow to write ("program")
- Nand-Flash, 1987
  - Fujio Masuoka
  - lowers the wiring per bit to one-eighth that of the Flash Memory's

Intel ETOX Flash

- Intel, 1997
- A single transistor with the addition of an <u>electrically isolated polysilicon floating gate</u> capable of storing charge (electrons)
- Negatively charged electrons act as a barrier between the control gate and the floating gate.
- Depending on the flow through the floating gate (more or less than 50%) it has value 1 or 0.
- Read/Write data in small blocks

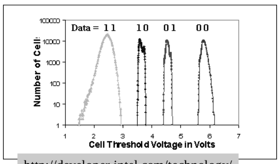
use high voltage to write, and "Fowler-Nordheim Tunneling" to clear



http://developer.intel.com/technology/itj/q41997/articles/art_1.htm

Intel StrataFlash

- Flash cell is analog, not digital storage
- Use different charge levels to store 2 bits (or more!) of data in each flash cell



http://developer.intel.com/technology/itj/q41997/articles/art_1.htm

Flash Implementations

- BIOS (PC's, phones, other hand-held devices....)
- Toshiba SmartMedia, 2-256 MB ....
- Sony Memory Stick, 2-256 MB
- CompactFlash, 8-512 MB ....
- PlayStation II Memory Card, 8 MB
- MMC - MultiMedia Card, 32-128 MB
- IBM MicroDrive (hard disk!) compatible memory card
- Hand-held phone memories

Cache Memory (välimuisti)

- Problem: how can I make my (main) memory as fast as my registers?
- Answer: (processor) cache
  - keep most probably referenced data in fast cache close to processor, and rest of it in memory
    - much smaller than main memory
    - (much) more expensive (per byte) than memory
    - most of data accesses to cache 90% 99%?

Fig. 4.3 & 4.6 (Fig. 4.13 & 4.16 [Stal99])

Memory references with cache (5)

- Data is in cache? Hit

Data is only in memory? Miss
    Read it to cache
    CPU waits until data available

Fig. 4.5 (Fig. 4.15 [Stall99])

<u>Many</u> blocks (cache lines) help for <u>temporal</u> locality
    many different data items in cache

Fig. 4.4 (Fig. 4.14 [Stall99])

<u>Large</u> blocks help for <u>spatial</u> locality
    lots of "nearby" data available

Fixed cache size?
    Select "many" or "large"?

Cache Features (6)

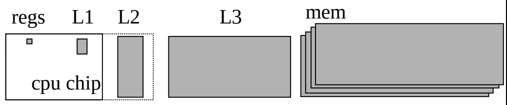
- Size
- Mapping function (kuvausfunktio)
  - how to find data in cache?
- Replacement algorithm (poistoalgoritmi)
  - which block to remove to make room for a new block?
- Write policy (kirjoituspolitiikka)
  - how to handle writes?
- Line size (block size)? (rivin tai lohkon koko)
- Number of caches?

12.9.2002 19

Cache Size

- Bigger is better in general
- Bigger may be slower
  - lots of gates, cumulative gate delay?
- Too big might be too slow!
  - Help: 2- or 3-level caches

1KW (4 KB), 128MW (512 MB)?



12.9.2002 20

Mapping: Memory Address (3)

- Alpha AXP issues 34 bit memory addresses
  - Use block address to locate block in cache
  - With cache hit, block offset is controlling a multiplexer to select right word

34 bit address (byte address)

block address	block offset
29 bits	5

Cache line size = block size = 2^5 = 32 bytes = 4 words

max physical address space = 2^{34} = 16GB

Number of possible blocks in physical address space = 2^{29} = 500M blocks

12.9.2002 21

Mapping (2)

- Given a memory block address,
  - is that block in cache?
  - where is it there?
- Three solution methods
  - direct mappings
  - fully associative mapping
  - set associative mapping

12.9.2002 22

Direct Mapping (6)

(suora kuvaus)

- Every block has only one possible location (cache line number) in cache
  - determined by index field bits

Block address (in memory)

34 bit address (byte address)

tag	index	block offset
21	8	5

Cache line size = block size = 2^5 = 32 bytes

Unique bits that are different for each block, stored in each cache line

Fixed addr in cache, cache size = 2^8 = 256 blocks = 8 KB

Fig. 4.7 (s = 29, r = 8, w = 5)
(Fig. 4.17 [Stall99])

Fig. 7.10 [PaHe98]

12.9.2002 23

Direct Mapping Example (5)

Word = 4 bytes (here)

ReadW I2, 0xA4

0xA4 = 1010 0100

8 bit address (byte address)

tag	index	offset
2	3	3
10	100	100

Cache line size = block size = 2^3 = 8 bytes = 64 bits

	tag (2)	data (64)
000:		
001:		
010:		
011:	01	54 A7 00 91 23 66 32 11
100:	11	77 55 55 66 66 22 44 22
101:	01	65 43 21 98 76 65 43 32
110:		

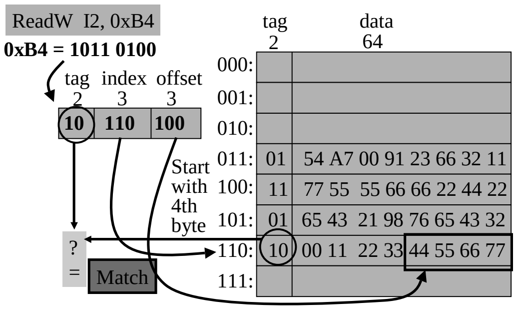
? =

No match

Read new memory block from memory address 0xA0=1010 0000 to cache location 100, update tag, and then continue with data access

12.9.2002

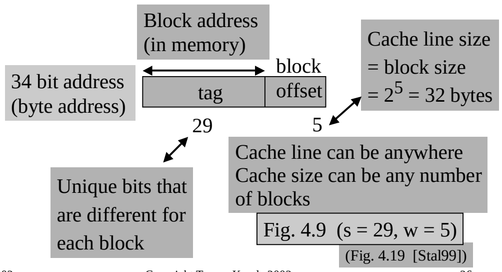
Direct Mapping Example 2 (5)



Fully Associative Mapping (5)

(täysin assosiatiivinen kuvaus)

- Every block can be in any cache line
  - tag must be complete block address



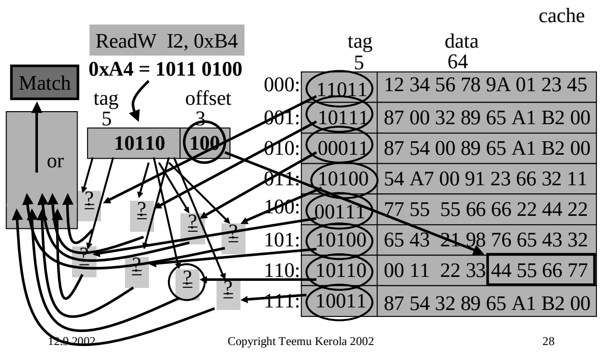
Fully Associative Mapping

- Lots of circuits
  - tag fields are long - wasted space!
  - each cache line tag must be compared simultaneously with the memory address tag
    - lots of wires
    - lots of comparison circuits

    Large surface area on chip
- Final comparison "or" has large gate delay
  - did any of these 64 comparisons match?

    $^2\log(64)$ = 8 levels of binary gates
  - how about 262144 comparisons? 18 levels?
- ⇒ Can use it only for small caches

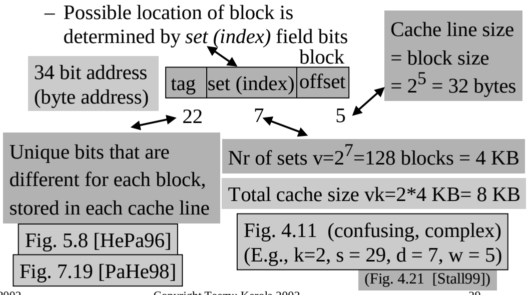
Fully Associative Example (5)



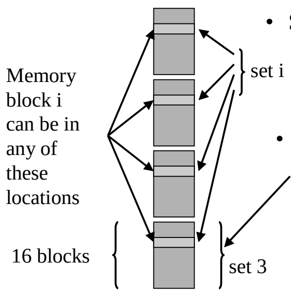
Set Associative Mapping (6)

(joukkoassosiatiivinen kuvaus)

- With set size k=2, every block has 2 possible locations in cache
  - Possible location of block is determined by *set (index)* field bits



Two definitions for "Set" in "Set Associative Mapping" (2)

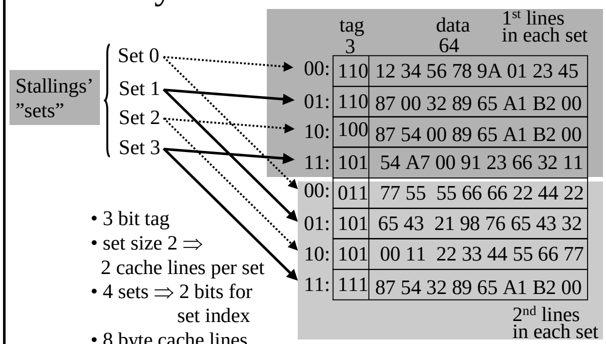


- Stallings "set"
  - "set" has all possible locations for block
  - 16 sets, set size 4
- Hennessy-Patterson "set"
  - each "set" has one possible location (index i) for block
  - 4 sets, set size 16

Two definitions for "Set" in "Set Associative Mapping"

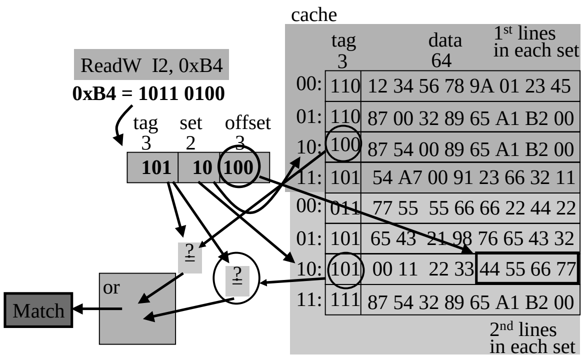
- Term "set" is the set of all possible locations where referenced memory block can be
  - Field "set" of memory address determines this set
  - [Stal03], [Stal99]
- Cache memory is split into multiple "sets", and the referenced memory block can be in only one location in each "set"
  - Field "index" of memory address determines possible location of referenced block in each "set"
  - [HePa96], [PaHe98]

2-way Set Associative Cache



- 3 bit tag
- set size 2 ⇒ 2 cache lines per set
- 4 sets ⇒ 2 bits for set index
- 8 byte cache lines ⇒ 3 bits for byte address in cache line

Set Associative Example (6)



Set Associative Mapping

- Set associative cache with set size 2 = 2-way cache
- Degree of associativity v? (Usually 2)
  - v large? (Fig. 7.16 [PaHe98])
    - More data items (v) in one set
    - less "collisions"
    - final comparison (matching tags?) gate delay?
  - v maximum (nr of cache lines) ⇒ fully associative mapping
  - v minimum (1) ⇒ direct mapping

Replacement Algorithm

- Which cache block (line) to remove to make room for new block from memory?
- Direct mapping case trivial
- First-In-First-Out (FIFO)
- Least-Frequently-Used (LFU)
- Random
- Which one is best?
  - Chip area?
  - Fast? Easy to implement?

Write Policy

- How to handle writes to memory?
- Write through (läpikirjoittava)
  - each write goes always to memory
  - each write is a cache miss!
- Write back (lopuksi kirjoittava takaisin kirjoittava?)
  - write cache block to memory only when it is replaced in cache
  - memory may have stale (old) data
  - cache coherence problem (välimuistin yhteneväisyysongelma)

Line size

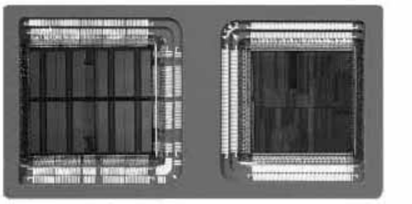
- How big cache line?
- Optimise for temporal or spatial locality?
  - bigger is better for spatial locality
- Data references and code references behave in a different way
- Best size varies with program or program phase
- 2-8 words?
  - word = 1 float??

Number of Caches (3)

- One cache too large for best results
- Unified vs. split cache (yhdistetty, erilliset)
  - same cache for data and code, or not?
  - split cache: can optimise structure separately for data and code
- Multiple levels of caches
  - L1 - same chip as CPU
  - L2 - same package or chip as CPU
    - older systems: same board
  - L3 - same board as CPU

Fig. 4.13
(Fig. 4.23 [Stal99])

-- End of Ch. 4-5: Cache Memory --



http://www.intel.com/procs/servers/feature/cache/unique.htm

“The Pentium® Pro processor's unique multi-cavity chip package brings L2 cache memory closer to the CPU, delivering higher performance for business-critical computing needs.“